V1.0 · NOVEMBER 2026

One function keeps the promise, its neighbour drops it

A documented argument, accepted and discarded by one method while its neighbour rejects it. Fifty-seven observations on whether a RAG stack applies the filter it was given.

Nikolaos Broikos · broikos.gr November 2026

57 observations · 19 broken filters · 24 Nov 2026

1 function

DECLARES KWARGS, DOCUMENTS THEM, DROPS THEM

19 of 19

DELIBERATELY BROKEN FILTERS CORRECTLY REFUSED

3 of 3

PREDICTIONS READ FROM SOURCE THEN CONFIRMED

4 to 19%

PER-DOCUMENT COVERAGE, THE HONEST WEAK POINT

01 One function keeps the promise, its neighbour drops it

> Discharges claims K001 to K014, K047.

A retrieval application serving more than one customer must answer a question on every query: which documents is this caller allowed to see. The usual answer is a metadata filter. Tag each document with an owner, pass the owner as a filter.

This paper asks one narrow thing about that answer. **When a RAG stack is given a per-owner metadata filter, is the filter applied, accepted and discarded, or rejected?**

Fifty-seven observations across Chroma, LanceDB, llama-index and two LangChain packages. The answer, in almost every cell, is that **the documented parameter is applied correctly.** Pass the name your layer declares and it works. There is one exception, it is specific, and it is three lines of somebody else's source.

The finding

Chroma.max_marginal_relevance_search, in langchain-chroma 1.2.0, declares **kwargs and documents them:

kwargs: Additional keyword arguments to pass to Chroma collection query.

It then calls the vector form without them:

```
embedding = self._embedding_function.embed_query(query)
return self.max_marginal_relevance_search_by_vector(
    embedding,
    k,
    fetch_k,
    lambda_mult=lambda_mult,
    filter=filter,
    where_document=where_document,
)
```

langchain_chroma/vectorstores.py, line 1075, and present on the project's master branch at the time of writing, not only in the release.

similarity_search in the same file carries **the identical docstring line** and does pass **kwargs. **max_marginal_relevance_search_by_vector passes them too. The outer MMR method is the only one of the three that does not, so** anything a caller puts there is discarded with no error and no warning.**

These are not inferences from reading. **An independent audit session read the same source and wrote all three as predictions, in a message to me, before any of them was run.** That independence is what makes them a control rather than a confirmation of my own expectation, and it is why they are attributed here rather than claimed.

A caution about the evidence for that ordering: predictions.json is written by the harness at the end of the run, so the file itself attests nothing about when the predictions were made. The ordering rests on the audit message, not on the file.

PREDICTION, WRITTEN BEFORE THE RUN	OUTCOME
max_marginal_relevance_search(filters=...) also discarded	discarded
max_marginal_relevance_search_by_vector(where=...) rejected	rejected
as_retriever(search_type="mmr", search_kwargs={"where": ...}) discarded	discarded

The second is the control that locates the defect. by_vector **does** forward kwargs, so the same argument collides there and Python raises. The gap is the outer method specifically, not "the MMR path" and not "LangChain".

The third matters most in practice, because the retriever is the interface applications are built on. langchain_core 1.8.4's VectorStoreRetriever passes search_kwargs straight into max_marginal_relevance_search, so an application that sets search_type="mmr" inherits the discard.

What that looks like from a caller's seat

Filtering for an owner that holds no documents, so an applied filter must return nothing whatever the ranking does:

CALL	RESULT
max_marginal_relevance_search(..., filter=nobody)	0 rows, applied
max_marginal_relevance_search(..., where=nobody)	**2 rows, discarded**
similarity_search(..., where=nobody)	TypeError, reaches the call
as_retriever(search_type="mmr", search_kwargs={"where": nobody})	**2 rows, discarded**

The same mistake raises on one method and fails open on the other.

Why a caller would write the other name

Because the ecosystem has three names for one idea and each is correct somewhere: Chroma calls it where, LangChain calls it filter, llama-index calls it filters. Someone who tests against Chroma directly and then moves behind a wrapper carries where with them.

This paper does not claim anyone does. No survey of real code was done, so the form is constructed and no statement about how often it occurs appears anywhere in this work.

Not a replication, and not two bugs

langchain-community 0.4.2 behaves the same way at chroma.py line 617. That is **the same function in two distributions of one code lineage, not an independent confirmation**, and it is reported that way throughout.

One difference matters for how the defect is described. langchain-chroma's docstring promises the kwargs reach the Chroma query, so there the code contradicts its own documentation. **langchain-community's MMR docstring does not mention kwargs at all**, so there the argument is undocumented and discarded, which is a weaker complaint. Three of the six discarded cells are from that package.

This is not a claim about access control

None of the libraries here claims to provide access control, and nothing in this paper tests whether they do. A metadata filter is a predicate the caller supplies. If the application builds it from request input, or lets a model write it, the caller's input shapes the filter and honouring it perfectly honours whoever shaped it.

This paper measures whether a supplied predicate is applied. **Tenant isolation is out of scope.** Section 05 is the binding version of that limit, and it applies to anyone quoting this work, including me.

Nor is it a vulnerability report. It is an API that discards a documented argument, and exposure requires an application to pass the wrong name to that one function.

It was reported upstream on the day of publication as langchain-ai/langchain#42065, scoped to langchain-chroma, which is open and labelled bug and chroma. The issue text and the reproduction, verified to run as written, are published with this paper.

The record is not tidy and is reported as it happened. A first attempt, #42064, was submitted through the REST API and **closed by a triage bot thirteen seconds later, because that project does not accept programmatic submissions.** Nothing in it was assessed. It was resubmitted through the web form as the bot asked, and the duplicate is disclosed in the new issue.

The upstream report is scoped to langchain-chroma alone, because the form requires confirming the issue is not about langchain-community. That package shows the same behaviour from the same code lineage, but its docstring does not promise the kwargs are forwarded, so the complaint there is weaker and it is not what was filed.

What behaved well, described rather than ranked

LanceDB refuses a misspelled field with a schema error that names the unknown field and lists the valid ones, and refuses a type mismatch rather than coercing it.

Chroma, natively, raises an explicit TypeError naming the unexpected argument for both filter= and filters=.

llama-index documents the Chroma-native where as a deliberate escape hatch, and raises ValueError if a caller supplies both filters and where, rather than letting one silently win. That guard was read in the source and then tested; it holds.

These are three different good behaviours and this paper does not rank them.

02 The instrument, and the control that makes it mean anything

> Discharges claims K015 to K021, K048.

The paper published the day before this one was retracted, because its generator and its checker implemented the same predicate and its headline measured neither. This instrument was designed against that failure, reviewed by an independent audit session before any data existed, and the design was fixed on disk first. PRE-REGISTRATION.md and call-forms.md are in the data package with the timestamps intact.

What is measured, and by whom

The thing under test is **the backend's behaviour**, which this paper does not write. The corpus generator assigns owner labels. It does not decide what a search returns. Ground truth and measurement therefore come from different code, owned by different people, and a wrong filter on my side shows up as a category, not as a silent pass.

Ground truth is the owner table, joined **by document id**. An owner value read back out of a search result is never trusted anywhere in the harness, because a store that drops or rewrites metadata would score a foreign document as not foreign.

The corpus

240 paragraphs drawn from 21 of 30 fetched English Wikipedia articles on maritime trade, CC BY-SA 4.0, with revision ids recorded in corpus-sources.json so the exact text is identifiable. Project Gutenberg was the first choice and refused automated fetches with HTTP 503 from two different addresses.

Four owners, 60 documents each, **assigned round robin over paragraph order**.

That assignment is load bearing and deliberate. If owners were topical, the embedding could encode owner, and a filtered query might look clean because the nearest neighbours happen to carry the right label. Round robin over one topical pool means every query retrieves paragraphs belonging to several owners, which is what allows the unfiltered control to leak, and makes the filter the only thing that can be doing the work. The build asserts no owner is more than 35% one source article. The largest actual share is **18.3%**: 11 of borealis's 60 paragraphs come from one article. An earlier draft said 13.8%, which was the largest share across the whole corpus and answered a different question than the one the assertion asks.

Four controls, all required per cell

1. **Positive.** The same query, through the same object and the same method, with only the filter removed, must return foreign documents in the top k. A raw client call does not stand in for the wrapper. 2. **Negative.** A filter for an owner with no documents must return empty. This was registered as required and **was missing from the first version of the bench**; DEVIATIONS.md records that and six other departures. 3. **Dialect recorded.** The exact call is logged per observation. 4. **Filtered recall, by document id.** The filtered result must account for the documents the same ranking owed it.

> Control 4 is the one that matters, and it is the one the first design lacked.

A filter on a misspelled field returns empty. Empty contains no foreign document, so it passes controls 1, 2 and 3 at once and scores as a clean enforcement. That is how a bench congratulates itself. Control 4 turns an empty or short result into unknown, which is a refusal to conclude rather than a pass.

It earned itself twice in this run, both times against this paper's own code rather than against a library:

- **LanceDB with prefilter=False returned 2 documents of the 10 expected**,

with no error. The filter is applied after the approximate nearest neighbour search, so legitimate documents are dropped for having fallen outside the unfiltered top k. Without control 4 that cell would have read as clean.

- **MMR with the correct filter scored unknown on the first run.** Section 08

records what that turned out to be and why the checker, not the library, was at fault.

Categories, fixed before the run

honoured · silently_ignored · rejected · unknown · path_dependent

rejected is counted as the **safe** failure and said so in the results. A library that raises on a filter it does not understand has done the right thing.

The harness tests itself

Nineteen degenerate forms run alongside the real ones: a misspelled field name, and a value of the wrong type, across seven distinct call paths. Seven, not six, because the retriever under search_type="similarity" and under search_type="mmr" are counted separately, and this paper's own finding is that they behave differently. Each must score unknown or rejected.

Per path, not per backend. The first version of this bench ran them per backend only, which left the relaxed control 4 of section 08 governing paths it had never been tested on.

If any of them scores honoured, the harness has proved it cannot tell a working filter from a broken one, and bench.py declares the run void and exits non-zero rather than publishing a table.

All nineteen scored unknown or rejected. LanceDB refuses a misspelled field with a schema error naming it and listing the valid fields, which is the behaviour this paper would want everywhere, stated as a description and not as a ranking.

What is not reported, and why

No leak rate. With a filter ignored, the share of queries that leak is set by the owner ratio, by k, and by topical overlap, all of which are parameters of this design. A percentage would be arithmetic over my own choices wearing the clothes of a measurement.

No count across libraries. The set is hand picked and the cells are not independent.

No query totals as n. An honoured equality filter is deterministic, so repeating a query ten thousand times is one observation of one code path, not ten thousand observations. The unit here is (wrapper, backend, call path, form) and there are 57 observations over 16 cells.

03 Every observation

> Discharges claims K022 to K026, K043 to K044.

Fifty-seven observations across sixteen cells. A cell is one (wrapper, backend, call path) at the versions in section 07, and each cell carries several filter forms. **No figure anywhere in this paper is per query**, for the reason in section 02.

The self check comes first, because it licenses everything after it

CALL PATH	BROKEN FILTERS RUN	SCORED UNKNOWN OR REJECTED
ChromaVectorStore.query	3	3
Table.search().where()	2	2
as_retriever.invoke	4	4
max_marginal_relevance_search	4	4
similarity_search	4	4
similarity_search_with_score	2	2

> 19 deliberately broken filters across 6 distinct call paths. 0 scored applied. A single one would void the run.

Nineteen filters were run that are **supposed** to fail: a field name absent from the index, and a value of the wrong type. They run on seven distinct call paths, including four on max_marginal_relevance_search and four on the retriever, which are the paths where the relaxed form of control 4 applies and therefore the paths where it most needed to be shown capable of failing.

None of the nineteen scored as applied. Had one done so, bench2.py exits non-zero and this paper does not exist, because a bench that cannot tell a working filter from a broken one reports its author's expectations.

The negative control ran on 56 of the 57 observations and no cell failed it.

The predictions, written from the source before the run

PREDICTION, WRITTEN FROM THE SOURCE BEFORE RUNNING	OUTCOME
max_marginal_relevance_search(filters=) silently ignored	**discarded**
max_marginal_relevance_search_by_vector(where=) rejected	rejected
as_retriever(search_type='mmr', search_kwargs={'where':}) silently ignored	**discarded**

These were read out of the code and recorded in predictions.json before anything was executed, so a confirmation cannot be reported afterwards as a discovery. The middle one is the control that locates the defect: the by_vector form **does** forward its keyword arguments, so the same argument collides there and raises. That is what makes the outer method, rather than MMR in general, the place where the argument is lost.

The six discards

PACKAGE	CALL PATH	ARGUMENT PASSED	ROWS RETURNED THAT A FILTER SHOULD HAVE WITHHELD
langchain-chroma over Chroma	max_marginal_relevance_search	where= chroma name	7, sample in the data capped at 5
langchain-chroma over Chroma	max_marginal_relevance_search	filters= llamaindex name	7, sample in the data capped at 5
langchain-chroma over Chroma	as_retriever.invoke, search_type mmr	search_kwargs where=, search_type mmr	7, sample in the data capped at 5
langchain-community over Chroma	max_marginal_relevance_search	where= chroma name	7, sample in the data capped at 5
langchain-community over Chroma	max_marginal_relevance_search	filters= llamaindex name	7, sample in the data capped at 5
langchain-community over Chroma	as_retriever.invoke, search_type mmr	search_kwargs where=, search_type mmr	7, sample in the data capped at 5

> One observation each. These are counts from a single call, not rates, and must not be rescaled or averaged: with four owners and k of 10 the unfiltered result is foreign-heavy by design.

Two call paths in each of two distributions of one code lineage, plus the retriever path on top of each. All six are the mechanism described in section 01: max_marginal_relevance_search accepts keyword arguments and does not pass them on. For the three langchain-chroma cells the docstring promises it will; **for the three langchain-community cells the docstring is silent**, so there the argument is merely undocumented rather than contradicted.

langchain_core 1.8.4's VectorStoreRetriever forwards search_kwargs into that method, which is why setting search_type="mmr" inherits the behaviour. **That is the path applications are actually built on**, and it is why this is worth a paper rather than a footnote.

Index state, which is where this kind of thing usually breaks

INDEX STATE	OUTCOME	NOTE
after a second add	applied	
after owner changed INTO subject	applied	absent from the top 10 only because it ranks 201st of 240 for this query; present at position 50 of 61 in the full filte
after owner changed back	applied	correctly stopped being returned once the owner was changed back
after deleting foreign documents	applied	no deleted document was returned

A filter verified on a freshly built static index tells you little, so these cells move the index underneath it: a second batch added after the first build, a document's owner changed into the subject and then back, and twenty foreign documents deleted.

Only the first of these four cells is informative, and this paper does not claim the other three as results. They were run, they returned applied, and an audit of the harness established that three of them could not have returned anything else. Reported here as what they are rather than quietly dropped:

- **The second add is informative.** Documents indexed after the first batch are

retrievable and the filter applies to them.

- **The owner-change cells are not.** The document whose owner was changed was

picked by corpus order, which happened to select one ranking 201st of 240 for this query. At k=10 it could never appear, whether the change was honoured or not. A separate check of the full filtered result found it at position 50 of 61 after the change, so Chroma did honour that update, but the cell as scored carries no weight, and **the mirror cell changing the owner back is uninformative for exactly the same reason.**

- **The delete cell cannot detect a failed delete.** Twenty *foreign* documents

were deleted and the query carried the *subject* filter, so a document that survived deletion would have been withheld by the filter anyway. To test a delete, subject documents must be deleted and then shown to stop returning.

The honest statement is therefore: **a filter applied correctly to documents added after the index was built, and the remaining index-state questions are open.** Section 06 names them as the next bench. The victim-selection rule is the fix: choose the documents to mutate from the unfiltered top k, not from corpus order.

Where behaviour was good, described and not ranked

LanceDB refuses an unknown field with a schema error that names it and lists the valid fields, and refuses a type mismatch rather than coercing it.

Chroma, natively, raises an explicit TypeError naming the unexpected argument for both filter= and filters=.

llama-index documents Chroma's native where as a deliberate escape hatch and **raises ValueError when a caller supplies both filters and where**, rather than letting one quietly win. That guard was read in the source and then tested; the cell confirms it.

Three different good behaviours, and this paper does not put them in an order.

One cell that is a correctness problem rather than a filtering one

LanceDB with prefilter=False returned 2 rows of the 10 requested, with no error. The filter is applied after the search rather than before it, so legitimate documents are dropped for having fallen outside the unfiltered candidate set.

A correction to how this was first described: no index was built on the table, so LanceDB ran an **exact** search here, not an approximate one. The post-filtering behaviour is the same, but the phrase "after the approximate nearest neighbour search" was wrong, and it also means this bench never tested LanceDB's indexed-against-unindexed-fragment behaviour at all. Nothing leaks. A caller gets a fifth of their results and no indication why, and the cell scores unknown rather than applied because a short result cannot demonstrate that a filter worked.

Every observation

CELL	CALL PATH	FILTER FORM	OUTCOME
langchain-chroma over Chroma	similarity_search	filter= correct	applied
langchain-chroma over Chroma	similarity_search	where= chroma name	rejected
langchain-chroma over Chroma	similarity_search	filters= llamaindex name	rejected
langchain-chroma over Chroma	similarity_search	filter= misspelled field *(broken on purpose)*	unknown
langchain-chroma over Chroma	similarity_search	filter= wrong type *(broken on purpose)*	unknown
langchain-chroma over Chroma	similarity_search_with_score	filter= correct	applied
langchain-chroma over Chroma	similarity_search_with_score	where= chroma name	rejected
langchain-chroma over Chroma	similarity_search_with_score	filter= misspelled field *(broken on purpose)*	unknown
langchain-chroma over Chroma	max_marginal_relevance_search	filter= correct	applied
langchain-chroma over Chroma	max_marginal_relevance_search	where= chroma name	**discarded**
langchain-chroma over Chroma	max_marginal_relevance_search	filters= llamaindex name	**discarded**
langchain-chroma over Chroma	max_marginal_relevance_search	filter= misspelled field *(broken on purpose)*	unknown
langchain-chroma over Chroma	max_marginal_relevance_search	filter= wrong type *(broken on purpose)*	unknown
langchain-chroma over Chroma	max_marginal_relevance_search_by_vector	where= chroma name	rejected
langchain-chroma over Chroma	max_marginal_relevance_search_by_vector	filter= correct	applied
langchain-chroma over Chroma	as_retriever.invoke	search_kwargs filter=	applied
langchain-chroma over Chroma	as_retriever.invoke	search_kwargs where=	rejected
langchain-chroma over Chroma	as_retriever.invoke	search_kwargs filter= misspelled *(broken on purpose)*	unknown

CELL	CALL PATH	FILTER FORM	OUTCOME
langchain-chroma over Chroma	as_retriever.invoke, mmr	search_kwargs where=, search_type mmr	**discarded**
langchain-chroma over Chroma	as_retriever.invoke, mmr	search_kwargs filter=, search_type mmr	applied
langchain-chroma over Chroma	as_retriever.invoke, mmr	search_kwargs filter= misspelled, mmr *(broken on purpose)*	unknown
langchain-community over Chroma	similarity_search	filter= correct	applied
langchain-community over Chroma	similarity_search	where= chroma name	rejected
langchain-community over Chroma	similarity_search	filters= llamaindex name	rejected
langchain-community over Chroma	similarity_search	filter= misspelled field *(broken on purpose)*	unknown
langchain-community over Chroma	similarity_search	filter= wrong type *(broken on purpose)*	unknown
langchain-community over Chroma	similarity_search_with_score	filter= correct	applied
langchain-community over Chroma	similarity_search_with_score	where= chroma name	rejected
langchain-community over Chroma	similarity_search_with_score	filter= misspelled field *(broken on purpose)*	unknown
langchain-community over Chroma	max_marginal_relevance_search	filter= correct	applied
langchain-community over Chroma	max_marginal_relevance_search	where= chroma name	**discarded**
langchain-community over Chroma	max_marginal_relevance_search	filters= llamaindex name	**discarded**
langchain-community over Chroma	max_marginal_relevance_search	filter= misspelled field *(broken on purpose)*	unknown
langchain-community over Chroma	max_marginal_relevance_search	filter= wrong type *(broken on purpose)*	unknown
langchain-community over Chroma	max_marginal_relevance_search_by_vector	where= chroma name	rejected
langchain-community over Chroma	max_marginal_relevance_search_by_vector	filter= correct	applied
langchain-community over Chroma	as_retriever.invoke	search_kwargs filter=	applied

CELL	CALL PATH	FILTER FORM	OUTCOME
langchain-community over Chroma	as_retriever.invoke	search_kwargs where=	rejected
langchain-community over Chroma	as_retriever.invoke	search_kwargs filter= misspelled *(broken on purpose)*	unknown
langchain-community over Chroma	as_retriever.invoke, mmr	search_kwargs where=, search_type mmr	**discarded**
langchain-community over Chroma	as_retriever.invoke, mmr	search_kwargs filter=, search_type mmr	applied
langchain-community over Chroma	as_retriever.invoke, mmr	search_kwargs filter= misspelled, mmr *(broken on purpose)*	unknown
llama-index over Chroma	ChromaVectorStore.query	filters= correct	applied
llama-index over Chroma	ChromaVectorStore.query	where= chroma name	applied
llama-index over Chroma	ChromaVectorStore.query	filters= AND where= together	rejected
llama-index over Chroma	ChromaVectorStore.query	filters= misspelled field *(broken on purpose)*	unknown
llama-index over Chroma	ChromaVectorStore.query	filters= wrong type *(broken on purpose)*	unknown
llama-index over Chroma	ChromaVectorStore.query	where= misspelled field *(broken on purpose)*	unknown
LanceDB, direct	Table.search().where()	where sql, prefilter=True	applied
LanceDB, direct	Table.search().where()	where sql, prefilter=False	unknown
LanceDB, direct	Table.search().where()	where sql, misspelled field *(broken on purpose)*	rejected
LanceDB, direct	Table.search().where()	where sql, wrong type *(broken on purpose)*	rejected
Chroma, direct, index mutated	Collection.query	after a second add	applied
Chroma, direct, index mutated	Collection.query	after owner changed INTO subject	applied
Chroma, direct, index mutated	Collection.query	after owner changed back	applied
Chroma, direct, index mutated	Collection.query	after deleting foreign documents	applied
Chroma, direct	Collection.query	where= correct, query 0	applied

Categories

OUTCOME	CELLS
applied	20
rejected	13
unknown	18
discarded	6

> Cells, not queries, and not a prevalence estimate: the libraries were chosen for running without a container, and a wrapper and its backend are not independent observations.

04 What to do about it

> Discharges claims K027 to K029, K045.

Four things follow from this bench. The first is small and immediate. The last is the one that matters, and it is not about any of these libraries.

1. Pass the name your layer declares

Three names exist for one idea and each is right somewhere: where in Chroma, filter in LangChain, filters in llama-index. **Every discarded cell involved passing a name that belongs to a different layer of the same stack.**

With one exception that should not be smoothed over: LanceDB's prefilter=False is a documented form, it was passed correctly, and it returned two rows of ten without an error. So "pass the documented name and it works" holds for the filter-name problem and not for everything.

So the narrow advice is: whichever layer your code touches, use that layer's own parameter, and never carry a name across a wrapper boundary because it worked one level down.

2. Assert on the result, not on the absence of an exception

The defect in section 01 produces no error. Neither does LanceDB's prefilter=False returning a fifth of the rows asked for. **A filter that silently does nothing and a filter that works are indistinguishable from the return value alone**, unless you check something about the rows.

The cheapest useful check, which is control 4 of this bench reduced to two lines:

```
rows = retriever.invoke(question)
assert rows, "no rows: the filter may have matched nothing"
assert all(r.metadata["owner"] == me for r in rows), "foreign row returned"
```

The second assertion is the one that would have caught the discard. The first is the one that catches the opposite failure, the filter matching nothing because of a typo or a type mismatch, which is the case this bench had to be redesigned to see at all.

Note that this check reads owner back out of the result, which this bench deliberately never does, because a store that drops metadata would make a foreign row look clean. In an application that trade is usually acceptable, since you are guarding against your own mistake rather than a hostile store. **It is not acceptable as the only guard if the threat model includes the store.**

3. Test the path you ship, not the path you learned on

similarity_search raises on a stray argument. max_marginal_relevance_search discards it. The retriever with search_type="mmr" discards it too, and the retriever is what applications are built on.

A filter verified against one method tells you nothing about another method of the same class. **If you switch to MMR for diversity, or set search_type, re-run whatever test convinced you the filter worked.**

4. A metadata filter is a convention, not a boundary

This is the one worth keeping.

Every cell in this bench puts all owners' documents in one collection and relies on a query-time argument to keep them apart. That argument is supplied by the caller, on every query, by hand. It is a convention held up by the discipline of whoever writes the next query, and section 01 shows what happens when a single keyword in it is wrong.

If the consequence of one wrong query is that one customer sees another customer's documents, the filter is not where that decision belongs. The alternatives, in rising order of cost and of confidence:

- **One collection per tenant.** The filter becomes a routing decision made once,

when the store handle is created, rather than a parameter repeated on every call. A wrong collection name fails loudly and empty rather than quietly and full. Nothing in this paper tested it, so this is a design argument and is typed as one in the ledger.

An earlier draft claimed this is what the ecosystem's own documentation recommends. **That was unsourced and it is contradicted at primary.** Qdrant's multitenancy guidance says a separate collection per tenant "is rarely the most efficient approach" and advises keeping tenants in one collection with payload partitioning, which is the pattern this section calls a convention. The trade-off is real in both directions and the vendors disagree about it.

- **One store or database per tenant**, where the backend supports it, which

moves the boundary to a layer that exists to enforce boundaries.

- **Filter where the identity is, not where the query is.** If the owner comes

from a verified session rather than from request input, a filter built from it is at least not attacker-shaped. **An honoured filter built from ?tenant= in a URL is honoured straight into the wrong tenant's documents**, and no amount of correct filtering fixes that.

What this paper does not tell you to do

It does not tell you to avoid any library here. Every one applied its own documented filter correctly in every cell where it had one, and the defect in section 01 requires passing an argument that layer never declared.

It does not tell you how common the mistake is. No survey of real code was done, so there is no prevalence claim anywhere in this work.

And it does not tell you that fixing this makes a system multi-tenant safe.

> Section 05 is the limit, and it is larger than the finding.

05 What this cannot show, and it is most of what you want to know

> Discharges claims K030 to K033, K046.

This section is binding on anyone quoting this work, including me. Its scope limits were set in the pre-registration before the results existed, because the time to decide what a result will not be allowed to mean is before you have it. The section itself was written afterwards and refers to the results, so it is not offered as a pre-registered document; PRE-REGISTRATION.md is.

A metadata filter is not access control

None of the libraries here claims to provide access control, and nothing in this paper should be read as testing whether they do.

A metadata filter is a predicate the caller supplies. The question this bench answers is whether a supplied predicate is applied. That is a necessary property and a boring one. It tells you nothing about the three places where multi-tenant retrieval actually fails:

Where the predicate comes from. If the application builds the filter from request input, the caller influences the filter. An honoured filter built from ?tenant= in a URL is honoured straight into the wrong tenant's documents. The filter working is not the question; who decided its value is the question.

Whether a model wrote it. SelfQueryRetriever lives in the langchain package, and langchain-community ships the ChromaTranslator that renders its output into a Chroma filter. An earlier draft said both wrapper packages ship self-query retrievers; langchain-chroma 1.2.0 ships none. Either way the pattern hands filter construction to a language model. Section 01 of paper 13 is the long argument for why a control that an attacker can write text into is not a control. A filter an LLM generates from attacker-influenced text is such a control, and a backend that honours it perfectly is honouring the attacker.

Whether the index is partitioned at all. Every cell here puts all owners' documents in one collection and filters at query time. That is the common pattern and it is why the question is worth asking, but a filter is a query-time convention, not a boundary. Nothing in this paper compares it against one collection per tenant, which fails

differently. The vendors do not agree on which is right: Qdrant's own multitenancy guidance recommends keeping tenants in a single collection with payload partitioning and calls per-tenant collections "rarely the most efficient approach".

> Tenant isolation is out of scope of this paper.

No prevalence claim

The libraries were chosen by hand, for one reason: they install and run on an embedded basis without Docker, which was the constraint of the machine available. That is a convenience sample. Chroma and LanceDB are here because they run in process; a product that needs a container is absent for that reason and not because it was judged.

A wrapper and the backend beneath it are not independent observations. Counting them as though they were would produce a number like "three of six leak", and that number would be meaningless.

> There is no figure in this paper for how common this defect is.

No library-level result generalises to a hosted service

qdrant-client 1.20.0 is in the data package but not in the results tables, and this is deliberate. **Its local mode is a separate pure-Python implementation under local/, written for testing, not the engine Qdrant runs.** A result from it would describe the emulator. Where it is mentioned at all it is labelled an emulator in the cell itself, rather than in a limitation a reader may not reach.

The same caution applies upward. Chroma and LanceDB running embedded are not Chroma Cloud and LanceDB Cloud. Nothing here was measured against a hosted service and nothing here predicts one.

No leak rate, and why a percentage would be dishonest

With a filter ignored, the proportion of returned documents belonging to other owners is determined by the number of owners, by k, and by how topically similar the corpus is. All three are parameters this paper chose. Four owners and k of 10 make roughly three quarters of an unfiltered result foreign, and that is arithmetic about the corpus design, not a finding about any library.

The counts in section 03 are reported as what one observation returned, never as a rate, and must not be rescaled, averaged or quoted as a probability. The leaked_sample field in results2.json is capped at five entries while n_leaked holds the true count, so that list is not a complete enumeration.

Index state is barely tested

Four cells moved the index underneath the filter, and **three of them turned out to be uninformative by construction.** Section 03 gives each one and why. The short version:

- **Tested:** documents added after the first batch was indexed. The filter

applied to them.

- **Not usefully tested:** a document whose owner was changed, in either

direction, because the document chosen ranked 201st of 240 and could not appear in a top-10 result either way. A separate check of the full result set showed the change was honoured, but the cell does not show it.

- **Not tested at all:** deletion. Foreign documents were deleted while the query

carried the subject's filter, so a document that survived deletion would have been withheld by the filter regardless.

Also untested: LanceDB's behaviour across indexed and unindexed fragments, because no index was ever built on the LanceDB table, so every LanceDB cell here ran an exact search.

A filter honoured on a near-static index may not be honoured during reindexing, and this paper does not know. Section 06 names it as the next bench.

One language, one filter shape

Every filter tested is a single equality on one string field. Conjunctions, negations, $in, numeric ranges and nested documents are untested, and a layer that honours an equality may translate a conjunction wrongly. The corpus is English. The owner values are ASCII.

06 Why this bench exists, and what the previous one taught

> This section argues from claims discharged elsewhere and introduces none of its own.

Two papers preceded this one, both published on 23 November 2026, the day before this one, and this one is the answer to what went wrong with the second.

Paper 13 tested whether machine-learned detectors can spot a prompt injection in an incoming email. Four detectors, three scoring modes. The finding was that the request invoice fraud actually uses, a polite note asking whoever drafts the reply to confirm a new bank account, passed every detector that kept ordinary mail clean. The argument that followed was that a control which must **recognise** an attack fails once somebody adapts to it, and that a **structural** control, one with no text to persuade, holds.

Paper 14 was meant to be the structural half of that argument and was retracted one day after publication. Its headline, that a deterministic comparison caught 160 of 160 fraudulent drafts, was a property of its corpus generator. There was no prose anywhere in its data, the generator defined each fraud class as putting a value in a field and the checker flagged a value not in a field, and the two fraud and legitimate bank-change classes were produced by the identical line of code. It reproduced byte for byte on every rerun and measured nothing.

So the lesson was not "be more careful". It was that **reproducibility is not validity**, and that the specific way to fail is to build an instrument whose answer is determined by code you wrote yourself.

What that changed about this bench

The thing under test is not mine. A backend's filtering engine decides what comes back. This paper's code assigns owner labels and asks questions. Ground truth and measurement are produced by different code owned by different people, which is the property paper 14 lacked entirely.

The design was fixed before the data existed. PRE-REGISTRATION.md was written and timestamped before the first run, including the list of things that would not be reported whatever the results showed: no leak rate, no count across libraries, no query totals as n.

An independent session was told to attack it. Not to review the finished paper, which is too late, but to attack the design while it was still a design. It found the defect that would have sunk this bench the same way paper 14 sank: my stated reason for believing three controls sufficient was **false**, because a filter on a misspelled field returns empty, and empty passes a leak test. That is control 4, and section 02 is where it earns itself.

The same session then audited the first run and found seven departures from my own pre-registration, three of them things I had simply not implemented. They are in DEVIATIONS.md with timestamps, including the one that is least comfortable: a control relaxed after seeing results.

The harness has to prove itself on every run. Nineteen deliberately broken filters, across six call paths. If any scores as applied, the run exits non-zero and there is no paper. Paper 14 had no such check, which is precisely why it could publish a tautology with full confidence.

The structural argument, and its honest status

Paper 13's argument survives this bench and is **not** demonstrated by it.

Comparing a request against a record the attacker cannot write to has no text to persuade, and that is a real structural advantage over reading the incoming message. A metadata filter is a thin version of the same idea: the owner value comes from your side, not from the document.

But section 04 is the limit on how far that goes. **A filter supplied by the caller on every query is a convention, and this bench has just shown that a single wrong keyword inside it fails open on one method and loudly on another.** A structural control that depends on a human writing the same argument correctly every time is only as structural as that habit.

Which is why the recommendation in section 04 is not "filter more carefully". It is to move the decision somewhere a mistake fails empty instead of full, and that recommendation is typed in the ledger as an **assumption**, because nothing here tested it.

What follows

Three things this bench could not reach, named rather than hinted at:

- **Filter shapes beyond a single equality.** Conjunctions, negations, $in and

numeric ranges. A layer that mistranslates a conjunction would be a worse version of section 01, because it would need no wrong parameter name.

- **Coverage.** Four to nineteen percent of foreign documents were ever retrieved

unfiltered, which bounds every applied result here. Only one cell ran more than a single query. More queries is the cheapest improvement available and it is the first thing to do.

- **Index state, properly.** Three of the four index-state cells here could not

have failed. Mutating documents chosen from the unfiltered top k, and deleting the subject's own documents rather than other owners', is the corrected design.

- **The hosted services.** Everything here runs embedded because no container

runtime was available. Chroma Cloud and LanceDB Cloud are untested and nothing in this paper predicts them.

07 Limitations

> Discharges claims K034 to K038, K049.

The honest summary of this paper's weight: **one defect, located precisely and reproduced from source, across six of fifty-seven observations, on a corpus whose coverage is low and whose scope is narrow.** The other fifty-one observations are not all clean behaviour: nineteen are filters broken on purpose, one is LanceDB's short result, and three are index-state cells that could not have failed.

Coverage is the weakest part

CELL	FOREIGN DOCUMENTS EVER RETRIEVED UNFILTERED	OF	COVERAGE
Chroma, direct	34	180	18.9%
langchain-chroma over Chroma	11	180	6.1%
langchain-community over Chroma	11	180	6.1%
Chroma, direct, index mutated	10	180	5.6%
llama-index over Chroma	8	180	4.4%
LanceDB, direct	8	180	4.4%

> A document never retrieved without a filter was never tested, so every applied in this paper is evidence about the documents these queries reached and not about the whole corpus. This is the weakest part of the bench and it is reported rather than omitted.

A document that never appeared in an unfiltered result was never available to leak, so it was never tested. Coverage ranges from 4.4% to 18.9% of the 180 foreign documents, depending on the cell.

This limits every applied in the paper, not the discards. A discard is positive evidence: rows came back that should not have. An applied means the filter held for the documents the queries reached, and says nothing about the other 80 to 95 percent.

Worse than the range suggests: **only one cell, Chroma direct, ran more than a single query.** The five queries exist in the harness, but every wrapper, llama-index, LanceDB and index-state cell used the first one, which is why those sit at 4.4 to 6.1 percent.

Raising coverage means more queries, which is cheap, and it is the first thing to do if any cell here is ever in doubt.

One filter shape

Every filter tested is a **single equality on one string field**. Untested: conjunctions, negations, $in, numeric comparisons, nested metadata, and null. A layer that translates an equality correctly may mistranslate a conjunction, and a translation bug there would be a more serious version of the finding in section 01 because it would not require a wrong parameter name.

One embedding model, one language, ASCII owners

all-MiniLM-L6-v2, English text, owner values that are plain ASCII. The corpus is 240 paragraphs from 21 English Wikipedia articles on maritime trade.

For an applied equality pre-filter the embedding cannot change the outcome, so this is not a validity limitation for most cells. It does bound the two cells where ranking matters, the MMR and prefilter=False ones.

Embedded backends only, and one library excluded

No container runtime was available on the measurement machine, so everything here runs in process. **Chroma and LanceDB running embedded are not Chroma Cloud and LanceDB Cloud**, and no result here predicts a hosted service.

qdrant-client 1.20.0 is installed and pinned but **absent from every results table on purpose.** Its local mode is a separate pure-Python implementation written for testing, not the engine Qdrant runs. A number from it would describe the emulator while appearing to describe Qdrant, which is the kind of figure this programme exists not to publish.

A convenience sample, so no prevalence claim

The libraries were chosen because they install and run without a container. That is a convenience sample, a wrapper and the backend under it are not independent observations, and **no count of the form "N of M leak" appears anywhere in this paper.**

Not two independent confirmations

langchain-chroma and langchain-community show the same behaviour from the same function in one code lineage. That is **one finding in two distributions**, not a replication.

The counts in section 03 are not rates

With four owners and k of 10, an unfiltered result is foreign-heavy by construction. The 7 in the discards table is what one call returned, and the leaked_sample field in results2.json is **capped at five entries** while n_leaked is the true count. Neither may be rescaled, averaged, or quoted as a probability.

Deviations from the registered design

Seven, including three that were my own omissions, listed in full with timestamps in DEVIATIONS.md. The two that most affect how much this paper is worth:

- **Control 4 was relaxed for re-ranking paths after seeing the first run.** The

original rule asked MMR to return the similarity ranking it exists to depart from. The relaxed rule now faces deliberately broken filters on the paths it governs and refuses them.

- **The first run's output was not preserved, and an earlier version of this paper

said it was.** results.json was overwritten by a later run of the same script, and the file published as results-run1-exact-control4.json is a byte-identical copy of that later output, already carrying the relaxed rule. It has been removed rather than left to look like evidence. The account of the first run in section 08 rests on the figures quoted there, not on a published artefact, and that is a weaker footing than this programme is supposed to accept.

- **The first pass ran one query per cell**, which is why coverage is reported

here as a limit rather than quietly omitted.

Versions, because any row here can move

PACKAGE	VERSION
chromadb	1.5.9
lancedb	0.41.0
langchain	1.5.2
langchain-chroma	1.2.0
langchain-community	0.4.2
langchain-core	1.8.4
llama-index-core	0.14.26
llama-index-vector-stores-chroma	0.7.0
llama-index-vector-stores-lancedb	0.7.0
numpy	2.5.5
python	3.12.3
qdrant-client	1.20.0
sentence-transformers	6.3.0

Both LangChain packages were the latest published releases on the day of the run, and the defect was confirmed present on the project's master branch, so it is not an artefact of an old pin.

08 What this bench got wrong about itself

> Discharges claims K039 to K042, K050.

Six times during this work the harness, or my account of it, reported something confidently that turned out to be wrong about itself. Four were found by an independent audit session rather than by me. Each is recorded here with what it produced and what it would have produced if unnoticed, because a programme that publishes only the errors it caught in other people's code is not being honest about where errors come from.

1. Control 4 was invalid for re-ranking paths

What it reported. max_marginal_relevance_search with the correct filter= argument scored unknown on both LangChain packages, flagged as "filtered result does not hold the documents this ranking owed it", with three or four documents missing.

What was actually true. The detail recorded ten documents returned of ten requested, every one belonging to the correct owner. The filter was honoured perfectly. The missing documents were missing because **MMR exists in order not to return the similarity top k**: it trades similarity for diversity among the results. Control 4 as first written demanded exactly the ranking MMR is designed to depart from.

The fix. Control 4 now splits. On a ranking-preserving path the returned set must equal what the ranking owed. On a re-ranking path the result must be full length and hold no foreign document, because the selection is deliberately not in similarity order. A short result still scores unknown on both, which preserves the case that matters.

What it would have cost. Two cells reported as suspicious that were correct, in a paper whose headline is about the same two methods. The temptation to report "MMR is unreliable with filters" was available and wrong.

How it was caught. By not believing the category and reading the detail underneath it. The counts, ten returned and ten asked for, contradicted the label.

2. A document id reader that returned None

What it reported. The langchain-community cell crashed with KeyError: None, taking out eight observations.

What was actually true. langchain-community's Chroma wrapper does not populate Document.id. The harness read d.id, got None, and used it as a key into the ground truth table.

The fix. Resolve ids by document text, which is unique in this corpus, with an assertion at load time that the texts really are unique so that this cannot rot silently. The owner is still never read from a result.

What it would have cost. A whole package silently absent from the results. A missing cell is a far more dangerous outcome than a wrong one, because nothing in a results table shows you the row that was never added.

3. The first design would have scored a broken filter as clean

This one was caught before any data existed, by the design review rather than by the run, and it is the most serious of the three.

The original design had three controls: an unfiltered query must leak, a filter for an empty owner must return nothing, and the exact call must be logged. The stated reason for believing it sound was that a wrong filter "shows up as an error or as a leak, not as a silent pass".

That reason is false. A filter on a misspelled field name returns an empty result. An empty result contains no foreign document, so it passes the leak test. It also passes the negative control, and the positive control passes independently of it. **All three controls go green on a filter that matched nothing at all.**

The fix is control 4 and the nineteen degenerate forms in section 03, which exist so the harness must demonstrate on every run that it can still tell a working filter from a broken one.

4. The self check was weaker than it was described as being

What it reported. "Eight degenerate forms, on every backend, all scoring unknown or rejected."

What was actually true. True per backend and **false per path.** No broken filter had been run on max_marginal_relevance_search or on the retriever, which are exactly the paths governed by the relaxed form of control 4 introduced in correction 1. So the relaxed rule had never been shown capable of failing anywhere it applied.

The fix. Nineteen degenerate forms across six distinct paths, four of them on MMR and four on the retriever, reported per path in section 03.

How it was caught. By the audit session reading the pre-registration against the code, not by me. Along with six other deviations in DEVIATIONS.md, including a control I had registered as required and never implemented.

5. "The first run is preserved unedited" was false

What it claimed. That a reader could inspect the stricter rule's verdicts in results-run1-exact-control4.json.

What was true. That file was a byte-identical copy of a **later** run, written four minutes after it, already carrying the relaxed rule. The original output had been overwritten. The file was evidence of nothing and it contradicted the claim it was offered to support.

The fix. The file is removed rather than left looking like evidence, this section and section 07 say the original is gone, and the ledger row for correction 1 is retyped as an assumption, because the artefact that would have made it own-data no longer exists.

How it was caught. By the audit session running cmp and reading the timestamps. Not by me, and this is the fourth time I have asserted provenance without opening the artefact I was pointing at.

6. Three of the four index-state cells could not have failed

What they reported. Four applied verdicts, written up as "Chroma honoured the filter in all four".

What was true. Only the first is informative. The document chosen for the owner-change cells was selected by corpus order and ranks 201st of 240 for the query, so it could not appear in a top-10 result whether the change was honoured or not. **This is the exact trap described in correction 1, repeated unnoticed in its own mirror cell.** The delete cell deleted *foreign* documents while querying with the *subject's* filter, so a document surviving deletion would have been withheld by the filter anyway.

The fix. Sections 03 and 05 now report one informative cell and three open questions, and section 06 names the corrected design: mutate documents drawn from the unfiltered top k, and delete the subject's own.

Why these are in the paper

The paper published the day before this one was retracted, one day after publication, because its generator and its checker implemented the same predicate and its headline measured neither. Its retraction notice is the first section of check-the-draft.html.

The lesson taken from it was not "be more careful". It was that **reproducibility is not validity**: that paper reproduced byte for byte on every rerun and was still empty. So this bench was pre-registered before it existed, reviewed by a session whose only instruction was to attack the design, and built to fail loudly on its own self check.

By the count that matters, this harness and its write-up got six things wrong and the libraries under test got one thing wrong. **That ratio is the normal one, and a paper reporting the opposite ratio should be read with suspicion.**

One more was caught in the results rather than the code. At k=10 the owner-change cell looked like Chroma ignoring a metadata update. It ranks 201st of 240 for that query and sits at position 50 of 61 in the full filtered result, so the update was honoured and the observation was uninformative. index-update-verification.json holds the check. **That would have been the easiest wrong finding in this paper**, and the only thing that stopped it was refusing to believe a category without reading the numbers under it.

ABOUT THE RECORD

Fifty-seven observations across Chroma, LanceDB, llama-index and two LangChain packages, each one a call path given a per-owner metadata filter. The corpus is 240 paragraphs of public Wikipedia text under CC BY-SA 4.0 with revision ids recorded. The harness, the pre-registration written before the first run, the deviations from it, and the upstream issue text are all published with the paper.

WHAT IT SHOWS

Every documented filter parameter was applied correctly. One method in two LangChain distributions declares keyword arguments, documents them as passed to the Chroma query, and discards them, so the same mistake raises an error on one method and fails open on another. The mechanism was predicted from the source by an independent audit session, written down as three predictions before the run, and then tested.

LIMITS AND CONFLICTS

Per-document coverage is 4 to 19 percent, which bounds every applied result though not the discards. Every filter is a single equality on one string field, everything runs embedded rather than hosted, and the library set is a convenience sample, so no prevalence claim is made and no leak rate is reported. Tenant isolation is outside the scope and no finding here licenses a claim about it. The author builds AI systems commercially and this paper argues for a design he would be paid to implement, which is why the bench is published in full. Unfunded, and the defect was reported upstream on the day of publication as langchain-ai/langchain issue 42065, scoped to langchain-chroma, after a first attempt through the API was closed by a triage bot for being submitted programmatically.

© 2026 Nikolaos Broikos. Text and data under CC BY 4.0, code under MIT. The data, code and claim ledger are in data/paper-15/ so any figure here can be recomputed, checked or disputed.